Decompositions of Natural Numbers: From A Case Study in Mathematical Theory Exploration *

Adrian Crăciun and Mădălina Hodorog
Institute e-Austria
Timişoara, Romania
{acraciun, mhodorog }@ieat.ro

Abstract

In the context of a scheme based exploration model proposed by Bruno Buchberger, we investigate the idea of decomposition, applied in the exploration of natural numbers. The free decomposition problem (i.e. whether an element can always be decomposed with respect to an operation) can be arbitrarily difficult, and we illustrate this in the theory of natural numbers. We consider a restriction, the decomposition in domains with a well-founded partial ordering: we introduce the notions of irreducible elements, reducible elements w.r.t. a composition operation, decomposition of domain elements into irreducible ones, and also the problem of irreducible decomposition which we then solve.

Natural numbers can be classified as a decomposition domain, in which we know how to solve the decomposition problem. This leads to the prime decomposition theorem.

Key words: *Systematic Theory Exploration, Knowledge Schemes, Prime Decomposition, Theorema.*

1 Introduction

The challenge of the computer playing a more important role as a practical tool for the working mathematician has been raised and is being addressed by research communities (see Calculemus [2], MKM [1]), building on results in automated reasoning, computer algebra, logic and mathematics, as well as recent computer technology development (communications, processing power).

As a contribution towards meeting this challenge - specifically providing computer support to the development of mathematical theories, Bruno Buchberger recently proposed a theory exploration model based on knowledge schemes, see [3, 4]. In this model, *knowledge schemes* respresent *abstractions of mathematical ideas*, and they are used in the development of mathematical theories.

The work presented in this paper is a case study of mathematical theory exploration using Buchberger's exploration model. We give an overview of the scheme based exploration model, with examples of the exploration process in the theory of natural numbers (Section 2), present the idea of free decomposition and its instantiations in natural number theory (Section 3), consider decomposition in domains with a well-founded partial ordering and the problem of irreducible decomposition in such domains (Section 4). Natural numbers can be classified as a decomposition domain, hence we have a solution to the irreducible decomposition problem for naturals, see Section 5. Section 6 mentions related work, while Section 7 contains concluding remarks and points to future work. The Appendix of this paper contains a list of knowledge schemes used.

Throughout this paper, we use the notational conventions of Theorema, see [4].

2 Systematic Exploration of Mathematical Theories

First, a remark concerning the language frame of our discussion. We use:

- *First order logic with equality*, together with the associated inference rules to express *mathematical facts*, and reason about them. The language is untyped, and (unary) predicates are used to express domain information ("types" or "sorts"). A fragment of this language can be used for programming: universally quantified formulae, with finite ranges, and recursion, using substitution and replacement as computational engines.

*Work supported by EU Marie Curie Project MERG-CT-2004-012718: SysteMaThEx.

- *Higher order logic* (with the restriction that higher order variables appear only universally quantified) are used to express *knowledge schemes*.

In particular, the THEOREMA language provides such a language frame.

Mathematical Theories: A mathematical theory $\mathcal{T}$ consists of:

- The *first order language* $\mathcal{L}$ describing the theory. It contains *predicate symbols* (including '=' and one or more unary predicates used to describe the "types" of objects), *function symbols* (including the identity) and *constant symbols*.
- The *knowledge base* $\mathcal{KB}$ of the theory (a collection of first–order formulae over the language $\mathcal{L}$).
- The *reasoning mechanism* $\mathcal{IR}$ of the theory (reasoning methods available to the developer of the theory, including first order predicate logic calculus and rewriting, plus theory-specific inference rules).

Example 1 (The Theory of Natural Numbers).

- $\mathcal{L}_{\mathbb{N}} = \left\langle \langle \textit{is-nat}, = \rangle, \langle {}^{+}, \textit{id} \rangle, \langle 0 \rangle \right\rangle$, where *is-nat* is the unary predicate symbol that characterizes natural numbers, $^{+}$ is an unary function symbol (the successor function), *id* is the identity (unary).
- $\mathcal{KB}_{\mathbb{N}}$ consists of the equality axioms, plus the well-known Peano axioms (generation, uniqueness and the induction principle). The induction principle cannot be formulated in first-order logic, therefore the axiom is lifted to the level of inference. In fact, the same happens to the substitutivity axioms for equality.
- $\mathcal{IR}_{\mathbb{N}}$ consists of first order predicate logic calculus, equality reasoning, and the induction inference rule.

For more details, see [7].

Knowledge Schemes: are higher order formulae that capture "interesting" mathematical knowledge.

Knowledge schemes are available at various levels of abstraction: some (e.g. algebraic structures, relations) are independent of mathematical theories, while others are not (see examples below).

Example 2 (Theory independent scheme).

$$\underset{p,op}{\forall} (\textit{is-semigroup}[p, op] \Leftrightarrow \underset{p[x,y,z]}{\forall} \bigwedge \left\{ \begin{array}{l} p[op[x, y]] \\ op[x, op[y, z]] = op[op[x, y], z] \end{array} \right.),$$

where p (unary "sort" predicates), op (binary functions) are higher order variables and *is-semigroup* is a special higher order constant (name, unique identifier of the scheme schemes). See also *is-monoid*, *is-group* in the Appendix.

Other theory independent schemes are relation structures, such as *is-preorder*, *is-partial-ordering*, *is-strict-partial-ordering*, see Appendix.

Example 3 (Theory-dependent schemes). Consider the theory of natural numbers, as introduced in *Example 2*. The following scheme captures the idea of a binary function defined recursively in terms of unary functions:

$$\underset{f,g,h}{\forall} (\textit{is-rec-nat-binary-fct-1r}[f, g, h] \Leftrightarrow \underset{\textit{is-nat}[x,y]}{\forall} \bigwedge \left\{ \begin{array}{l} f[x, 0] = g[x] \\ f[x, y^{+}] = h[f[x, y]] \end{array} \right.) .$$

The recursive structure of the above scheme reflects the inductive structure of natural numbers. In fact, a whole range of schemes can be constructed using the inductive structure of a particular domain, see the Appendix for more examples.

Exploration: An *exploration situation* consists of a theory being developed, together with a library of knowledge schemes.

The basic steps that can be taken in the development of a theory $\mathcal{T} = \langle \mathcal{L}, \mathcal{KB}, \mathcal{IR} \rangle$ are:

- Add concepts to the language and explore their properties. *Concepts* in mathematical theories are functions, predicates or constants. Mathematical theories are augmented with the introduction of new functions or predicates, by definition. *Definition schemes* can be instantiated with a new symbol (which is added to the language $\mathcal{L}$), for the concept being defined, and with symbols from $\mathcal{L}$ corresponding to the concepts used to define the new concept.

- Add propositions to the knowledge base. A first step in the development of a knowledge base attached to a theory is to derive "interesting, useful" consequences of the axioms/definitions: modify the statements of the axioms/definitions by exploiting their structure (e.g. if an axiom about left neutral element is present in $\mathcal{KB}$, try to prove the right neutral property), or derive consequences by forward reasoning.

Then investigate *structural properties of the concept* (commutative, associative, etc. for binary functions, equivalence or an ordering for binary predicates, etc.), which are proposed by instantiation of *proposition schemes*.

The aim of this type of exploration round is to "saturate" $\mathcal{KB}$, i.e. to add "all interesting" consequences of an initial knowledge base. This is, however, necessarily an incomplete process (in most of the cases) and could lead to an explosion in the number of propositions.

- Solve problems. *Problems* can be introduced in the theory by instantiations of *problem schemes*. A method for solving algorithmic problems in the context of scheme-based systematic theory exploration, synthesis by *lazy thinking*, was proposed by Bruno Buchberger, see [4]. It consists of the following:

- a solution for the problem is proposed by selection of an *algorithm scheme* available,
- this is instantiated with new symbols (not present in the language) and a correctness proof is set up,
- the proof will very likely fail, due to the new symbols,
- the failed proof is analysed and conjectures on the new symbols are formulated, that allow the proof to get over the failure,
- the conjectures are specifications (problems) of the new symbols, and either retrieve concepts that verify the specifications, or the synthesis process is repeated.

- Add new inference rules to the inference mechanism. The way to add new inference rules to $\mathcal{IR}$ is to *lift knowledge to the level of inference*. This is a process very common in mathematics. As a result of lifting knowledge to inference, $\mathcal{IR}$ and $\mathcal{KB}$ are updated.

Developing Natural Numbers Theory in The Scheme Based Exploration Model. We already presented details of the initial development of the theory of natural numbers in [7]. We now give a "trace" ("script") in the form of scheme instantiations, for this development.

Introducing function symbols: *is-rec-nat-binary-fct-1r*$[+, id, {}^{+}]$ generates the definition for the usual addition, $+$. Instantiations of knowledge schemes generate properties of $+$:

- *is-rec-nat-binary-fct-1l*$[+, id, {}^{+}]$ (an equivalent definition);
- *is-semigroup*$[$*is-nat*$, +]$ (i.e. sort and the associativity properties);
- *is-monoid*$[$*is-nat*$, +, 0]$ (in addition, the neutral element property, which is already part of the definition);
- none of the possible instantiations for the *is-group* knowledge scheme can be proved (*is-group*$[$*is-nat*$, +, 0, id]$, *is-group*$[$*is-nat*$, +, 0, {}^{+}]$). We then attempt to solve the problem of finding an inverse for natural numbers, i.e. invent $\ominus$ such that *is-group*$[$*is-nat*$, +, 0, \ominus]$. We apply the lazy thinking synthesis method which gives that no such function exists, see [7].

In the same way, using the knowledge schemes we generate new notions and their properties: the multiplication $*$, the exponentiation $^{\wedge}$ and the predecessor $^{-}$ function symbols.

Introducing predicate symbols: The weak less-than relation $\leq$ is given by *is-rec-nat-binary-rel-2*$[\leq, id, {}^{+}]$. *is-rec-nat-binary-rel-2*$[<, {}^{+}, id]$ gives the strict variant $<$. Properties of $<$ are obtained from further instantiations:

- *is-strict-partial-ordering*$[$*is-nat*$, <]$ (i.e. the irreflexivity, the transitivity and the asymmetry properties);
- *is-strict-total-ordering*$[$*is-nat*$, <]$ (the totality property).

Introducing a new inference mechanism: With the strict-less than predicate symbol ($<$) we can introduce a new inference mechanism, the complete induction principle, and prove it is correct, see [7].

More Examples: The complete induction rule allows us to write new recursive knowledge schemes, which will generate new notions and their properties: the subtraction ($-$), the quotient, the remainder, and the greatest common divisor function symbols; *is-nat-step-recr-pred-0-1-1*$[|, False, id_B, True]$ generates the definition of divides $|$ (id_B is the identity function on the boolean domain). Properties of $|$:

- *is-preorder*$[$*is-nat*$, |]$ (i.e. the reflexivity and the transitivity properties);
- *is-partial-ordering*$[$*is-nat*$, |]$ (i.e. the antisymmetry property).

Because $|$ is a weak partial ordering, we introduce in the language its strict version (proper divides, $\lhd$).

3 Free Decomposition Problem

We describe here the idea (scheme) of *free*[1] *(binary) decomposition*, i.e. whether in a domain any element can be decomposed into two others, w.r.t. a binary operation, and free decomposition with pivot, i.e. whether any element can be decomposed into a given element and something else, w.r.t. a binary operation.

The free decomposition knowledge scheme:

$$\underset{obj,p_1,p_2,\otimes}{\forall}(FD[obj,p_1,p_2,\otimes] \Leftrightarrow \underset{\substack{obj[x]obj[y,z]\\ p_1[x]\ p_2[y,z]}}{\forall}\ \underset{}{\exists} x = y \otimes z).$$

The knowledge scheme for binary free decomposition

[1] *Free* refers to the fact that the operation used in the decomposition, as well as the predicates characterizing the variables can be chosen arbitrarily.

with pivot:

$$\underset{obj,p_1,p_2,p_3,\otimes}{\forall}(FBDp[obj,p_1,p_2,p_3,\otimes] \Leftrightarrow \underset{\substack{obj[x,y]\\ p_1[x],p_2[y]}}{\forall}\ \underset{obj[z]\ p_3[z]}{\exists} x = y \otimes z).$$

It turns out that instantiations of free decompositions can lead to arbitrarily complex problems, e.g.:

$$FD[\textit{is-nat}, \textit{is-odd-greater-2}, \textit{is-prime}, +] \Leftrightarrow \underset{\substack{is\text{-}nat[x]\\ is\text{-}odd[x]\wedge x>2}}{\forall}\ \underset{is\text{-}nat[y,z]\ is\text{-}prime[y,z]}{\exists} x = y + z,$$

which is the Goldbach conjecture.

The free decomposition with pivot is potentially easier to handle, e.g.:

$$FBDp[\textit{is-nat}, \textit{true}, \textit{is-positive}, \textit{true}, *] \Leftrightarrow \underset{\substack{is\text{-}nat[x,y]\\ is\text{-}positive[y]}}{\forall}\ \underset{is\text{-}nat[z]}{\exists} x = y * z.$$

In the right formula of this scheme instantiation, we apply skolemization to eliminate the existential quantifier, which leads to the following problem:

$$\underset{\substack{is\text{-}nat[x,y]\\ is\text{-}positive[y]}}{\forall} x = y * q[x,y],$$

i.e. whether any natural number x can be decomposed into a positive y and something else.

Applying the lazy thinking method, we attempt to "invent" the appropriate function q that solves this problem. The problem turns out to be unsolvable (following certain proof failures during lazy thinking), but a modification of this problem (suggested by these failures) can be solved:

$$\underset{\substack{is\text{-}nat[x,y]\\ is\text{-}positive[y]}}{\forall} x = y * q[x,y] + r[x,y]$$

with the solution being the usual quotient and remainer functions for naturals (the *quotient remainder theorem*). The details of the lazy thinking synthesis can be found in [6, 7].

4 Decomposition w.r.t. Well–Founded Orderings

In this section, we introduce the idea of decomposition into irreducible elements, w.r.t. a well-founded partial ordering. We give the corresponding schemes, and derive some properties of these schemes.

Well Founded Partial Orderings, Minimal Elements:

$$\underset{<,e,obj}{\forall}(\textit{is-well-founded}[<,e,obj] \Leftrightarrow \bigwedge \left\{ \begin{array}{l} \textit{is-strict-partial-ordering}[<,obj] \\ \textit{is-minimal-element}[e,<,obj] \end{array} \right.),$$

$$\underset{e,<,obj}{\forall}(\textit{is-minimal-element}[e,<,obj] \Leftrightarrow (obj[e] \bigwedge \underset{\substack{obj[x]\\ x<e}}{\forall} e < x)) .$$

The above schemes capture the idea of a well-founded partial ordering $<$, with a minimal element e, on a domain described by obj.

Compatible Composition: We now introduce the idea of a *compatible composition operation*, i.e. if a nontrivial object is the result of composition of two other nontrivial objects (w.r.t. a well-founded partial ordering), this operation is compatible with the ordering if the composed object is "larger" than its components:

$$\underset{\bigotimes,<,e,obj}{\forall}(\textit{is-compatible-composition}[\bigotimes,<,e,obj] \Leftrightarrow \bigwedge \left\{ \begin{array}{l} \textit{is-well-founded}[<,e,obj] \\ \underset{\substack{obj[x,y,z]\\ e<x,e<y,e<z}}{\forall} ((x = y \bigotimes z) \Rightarrow y < x \wedge z < x) \end{array} \right.).$$

Irreducible Elements w.r.t. a Well Founded Partial Ordering: For a domain with a well-founded partial ordering, the scheme introducing the predicate for irreducible elements is:

$$\underset{P,<,e,obj}{\forall}(\textit{is-irreducible-property}[P,<,e,obj] \Leftrightarrow \bigwedge \left\{ \begin{array}{l} \textit{is-well-founded}[<,e,obj] \\ \underset{\substack{obj[x]\\ e<x}}{\forall} P[x] \Leftrightarrow \underset{\substack{obj[y]\\ e<y}}{\forall} y \not< x \end{array} \right.).$$

In the same time, in a domain with a compatible composition w.r.t. the partial ordering, we can also formulate the scheme introducing the predicate for elements that are composed:

$$\underset{is\text{-}comp,<,e,\bigotimes,obj}{\forall}(\textit{is-comp-prop}[\textit{is-comp},\bigotimes,<,e,obj] \Leftrightarrow \bigwedge \left\{ \begin{array}{l} \textit{is-compatible-composition}[\bigotimes,<,e,obj] \\ \underset{\substack{obj[x]\\ e<x}}{\forall} \textit{is-comp}[\bigotimes][x] \Leftrightarrow \underset{\substack{obj[y],obj[z]\\ e<y,e<z}}{\exists} x = y \bigotimes z \end{array} \right.).$$

A Composed Object is not Irreducible: It should be obvious that the two ideas introduced above are connected: elements that are composed are not irreducible. Since schemes are higher order formulae, we

use the "arbitrary but fixed" rule to prove their properties.

Assume, on a domain described by obj, with predicates $<$, P, *is-comp*, function $\bigotimes$, constant e (now arbitrary but fixed constants), that:

$$\begin{array}{l} \textit{is-well-founded}[<, e, obj], \\ \textit{is-irreducible-property}[P, <, e, obj], \\ \textit{is-comp-prop}[\textit{is-comp}, \bigotimes, <, e, obj]. \end{array}$$

Then, $\underset{\substack{obj[x] \\ e<x}}{\forall}(\textit{is-comp}[\bigotimes][x] \Rightarrow \neg P[x])$.

We skip the proof here (it is simple, by predicate logic).

Decomposition Domains: are domains with a well-founded partial ordering, a compatible operation, s.t. any nontrivial reducible element is the composition of two nontrivial elements:

$$\underset{\otimes,<,e,obj}{\forall}(\textit{is-decomposition-domain}[obj, \otimes, P, <, e] \Leftrightarrow \bigwedge \left\{ \begin{array}{l} \textit{is-compatible-composition}[\otimes, <, e, obj] \\ \textit{is-irreducible-property}[P, <, e, obj] \\ \underset{\substack{obj[x] \\ \neg P[x] \wedge e<x}}{\forall} \underset{\substack{obj[y,z] \\ e<y \\ e<z}}{\exists} x = y \otimes z \end{array} \right.).$$

Irreducible Decomposition: For a domain with irreducible elements and a compatible composition operation, the scheme for introducing the notion of decomposition into irreducible elements is:

$$\underset{\textit{is-D},obj,<,e,\otimes,P}{\forall}(\textit{is-irred-decomp}[\textit{is-D}, <, e, \otimes, P] \Leftrightarrow \bigwedge \left\{ \begin{array}{l} \textit{is-compatible-composition}[\otimes, <, e, obj] \\ \textit{is-irreducible-property}[P, <, e, obj] \\ \underset{\substack{obj[x] \\ e<x}}{\forall} \bigwedge \left\{ \begin{array}{l} P[x] \Rightarrow \textit{is-D}[\lfloor x \rfloor, x] \\ \neg P[x] \Rightarrow \\ \quad \underset{\substack{obj[y], \textit{is-ms}[obj][D] \\ P[y]}}{\forall} \textit{is-D}[y \odot D, x] \Leftrightarrow \\ \quad \bigwedge \left\{ \begin{array}{l} y < x \\ \textit{is-D}[D, x \ominus y] \end{array} \right. \end{array} \right. \end{array} \right.),$$

where $\textit{is-ms}[obj][D]$ represents the multiset D of objects obj (we use $\lfloor\ \rfloor$ to denote a multiset), $\odot$ represents the insertion operation for multisets, and $\ominus$ represents the division operation for objects obj.

The Problem of Irreducible Decomposition: We want to solve the problem of decomposition into irreducible elements:

$$\underset{\substack{obj[x] \\ e<x}}{\forall} \textit{is-D}[Dec[x], x],\ (\diamondsuit)$$

that is for any obj x, with $e < x$, find an algorithm Dec such that the *is-D* problem defined by the $\textit{is-irred-decomp}[\textit{is-D}, <, e, \otimes, P]$ scheme holds.

For solving this problem we propose the following algorithm:

$$\begin{array}{l} \textit{Algorithm}[\text{"}\textit{decomposition}\text{"}, \textit{any}[obj[x], obj[y], \\ \qquad \textit{is-list}[obj][Z]], \textit{with}[P[y]], \\ Dec[x] = Dec[x, \textit{list-Irred}[x]] \\ Dec[x, \langle\ \rangle] = \lfloor\ \rfloor \\ Dec[x, y \smile Z] = \left\{ \begin{array}{l} y \odot Dec[x \ominus y, y \smile Z] \Leftarrow y < x \\ Dec[x, Z] \Leftarrow otherwise \end{array} \right. \end{array}],$$

where $\textit{list-Irred}[x]$ is a function that gives the list of all irreducible elements, up to x, where "up to" means that our well-founded ordering $<$ can be embedded in some other well-founded ordering, and the algorithm can generate all the irreducible elements up to x (like the sieve of Eratosthenes, for natural numbers).

It is easy to see, that, in fact, if we make the above assumption about $\textit{list-Irred}[x]$, with x not irreducible, by an easy proof, the irreducible elements y from the list such that $y \not< x$, do not influence the result, and are just discarded (branch "otherwise" in the algorithm above). If x is irreducible, then the list contains only x itself. In the following we assume this assumption for *list-Irred*.

Correctness of the Irreducible Decomposition Algorithm: The algorithm terminates (the binary version of the function Dec terminates). This is due to a lexicographic ordering ($<$ for the first argument and the length of lists for the second).

We prove by well–founded induction w.r.t. $<$ that

$$\underset{\substack{obj[x] \\ e<x}}{\forall} \textit{is-D}[Dec[x], x].\ (\diamondsuit)$$

Proof. Take arbitrary but fixed x_0 such that $obj[x_0]$ and $e < x_0$.

Assume $\underset{\substack{obj[y] \\ e<y}}{\forall}(y < x_0 \Rightarrow \textit{is-D}[Dec[y], y])$, (1)

Show $\textit{is-D}[Dec[x_0], x_0]$.

Case $P[x_0]$:

By definition of *is-irreducible-decomp*, we have to prove $\textit{is-D}[Dec[x_0, \langle x_0 \rangle], x_0]$.

Using the multisets property $\langle x_0 \rangle = x_0 \smile \langle\rangle$, we have to prove:

$$\textit{is-D}[Dec[x_0, x_0 \smile \langle\rangle], x_0].$$

By definition of *Dec*, we have to prove $\textit{is-D}[x_0 \odot Dec[x_0 \ominus x_0, \langle\rangle], x_0]$, i.e. $\textit{is-D}[x_0 \odot \langle\rangle, x_0]$. By the multisets properties, it follows $\textit{is-D}[\lfloor x_0 \rfloor, x_0]$.

Case $\neg P[x_0]$:

We have to show *is-D*[*Dec*[x_0, *list-Irred*[x_0]], x_0].

Case *list-Irred*[x_0] = $\langle\rangle$:

From the specifications of *list-Irred* follows

$$\overset{\nexists}{\underset{P[y]}{\underset{obj[y]}{}}} y \lhd x_0.$$

Therefore, necessarily, $x_0 = e$, which is false, so formula ($\diamondsuit$) holds.

Case *list-Irred*[x_0] = $y_0 \smile Z_0$:

We have to show *is-D*[*Dec*[$x_0, y_0 \smile Z_0$], x_0].

Case $y_0 < x_0$:

By definition of *Dec*, we have to show *is-D*[*Dec*[$x_0 \ominus y_0, y_0 \smile Z_0$], $x_0 \ominus y_0$].

But $x_0 \ominus y_0 < x_0$, so from (1) we know *is-D*[*Dec*[$x_0 \ominus y_0$], $x_0 \ominus y_0$], i.e.

$$\textit{is-D}[\textit{Dec}[x_0 \ominus y_0, \textit{list-Irred}[x_0 \ominus y_0]], x_0 \ominus y_0]. \ (\triangle)$$

Only two cases are possible (from the specifications of *list-Irred*).

If $y_0 \lhd x_0 \ominus y_0$, then *list-Irred*[$x_0 \ominus y_0$] = $y_0 \smile Z_0$, and we are done.

If $y_0 \not\lhd x_0 \ominus y_0$, then *list-Irred*[$x_0 \ominus y_0$] = Z_0.

From the definition of *Dec* (*otherwise* branch)

$$\textit{Dec}[x_0 \ominus y_0, y_0 \smile Z_0] = \textit{Dec}[x_0 \ominus y_0, Z_0].$$

With this the goal follows immediately from formula ($\triangle$).

Case $x_0 < y_0$: The assumptions on *list-Irred* guarantee that this case is not needed.□

Uniqueness of the Irreducible Decomposition:

We have just shown that the irreducible decomposition problem has a solution (given by the algorithm *Dec*). In fact, from the definition of irreducible decomposition, it follows immediately that if we have two decompositions, these have to be equal: indeed, for any irreducible element y in one decomposition of x, we have $y < x$, that is, it necessarily has to be in any decomposition of x. The proof works by induction on multisets.

Summary (Irreducible Decomposition in Decomposition Domains):

We have introduced an abstract notion of irreducible elements w.r.t. a well-founded partial ordering, and then the problem of decomposition into irreducible elements on such domains, proposed a solution (provided we can find an enumeration - *list-Irred* - of irreducible elements) and proved the solution is correct.

Note that this algorithm is not the only solution: a *divide-and-conquer* scheme can also be applied, which would lead to a proof similar to the one presented in [9], for decomposition of natural numbers. However, it is easy to prove that whatever the method, the result is unique. We can now use the results we obtained in any decomposition domain, i.e. the problem of irreducible decomposition is reduced to the problem of classifying a domain as a decomposition domain.

5 Prime Decomposition of Natural Numbers

Natural Numbers Form A Decomposition Domain:

We prove:

is-decomposition-domain[*is-nat*, $*$, *is-prime*, $\lhd$, 1], i.e.

$$\bigwedge \begin{cases} \textit{is-compatible-composition}[*, \lhd, 1, \textit{is-nat}] & (1) \\ \textit{is-irreducible-property}[\textit{is-prime}, \lhd, 1, \textit{is-nat}] & (2) \\ \underset{\textit{is-nat}[x]}{\forall} \ \underset{\textit{is-nat}[y,z]}{\exists} x = y * z & (3) \\ \neg \textit{is-prime}[x] \wedge 1 \lhd x \ \ 1 \lhd y, 1 \lhd z & \end{cases}.$$

Formula (1) is proved by contradiction using the properties of $|$ and $\lhd$. Formula (2) holds because it represents the definition for the *is-prime* predicate symbol. Formula (3) is proved using the *quotient-remainder* theorem with the divisibility properties (which already provides the decomposition).

Irreducible Decomposition of Natural Numbers:

For the decomposition domain determined above, the instantiation *is-irred-decomp*[*is-D*, *is-nat*, $\lhd$, 1, $*$, *is-prime*] gives the definition for the irreducible decomposition of natural numbers. The problem of decomposition (which for natural numbers is the prime decomposition theorem[2]) has a solution, provided by the appropriate instantiation of the *Dec* algorithm. In particular, it is easy (although not efficient, perhaps), to enumerate the list of primes up to an element.

Another irreducible decomposition is possible for the instantiation *is-irred-decomp*[*is-D1*, *is-nat*, $<$, 0, $+$, *is-irred-plus*]. The decomposition problem states that any natural number can be decomposed in an unique way in sum of irreducible elements w.r.t. $+$, i.e. in sum of 1's.

6 Related Work

Other systems are also concerned with the automated invention of mathematical concepts and theorems from the number theory.

[2]Primes and irreducible elements coincide for natural numbers.

AM program, see [8], re-invented some concepts from the number theory: the prime numbers, the highly composite numbers, the fundamental theorem of arithmetic, and Goldbach's conjecture. AM started with elementary concepts as sets and bags, and used certain heuristics to produce conjectures involving the concepts. Even though AM rather used a theorem proposer than a theorem prover, the program offered a methodology for computer-supported invention of mathematical concepts.

Starting with few definitions, HR represents mathematical concepts as data-tables and invents new mathematical notions from old ones using certain production rules, see [5]. Some results of the HR program in number theory are: the re-invention of the prime numbers and square numbers, the invention of the refactorable numbers (the integers for which the number of divisors is itself a divisor).

MATHsAID generates automatically mathematical concepts, starting with an initial set of axioms and then filters the interesting theorems, see [10].

7 Discussion

Summary. We presented a fragment from a case study of exploration in the theory of natural numbers, using a scheme based theory exploration model. We presented the model, an overview of the first stages of exploration and we investigated how various versions of the idea of decomposition can be used in our exploration. In fact, we explore the notion of decomposition at an abstract level, where we solve the problem of decomposition in decomposition domains. A classification of natural numbers as decomposition domain leads to an immediate solution for the irreducible (prime) decomposition of natural numbers.

Exploration vs. Formalization. We want to make clear that what we are doing is *exploration*, i.e. different users can choose different exploration paths ("scripts"), where every step is completely formal[3]. The theory is developed by instantiation of schemes, in their various roles (definitions, properties, problems). Exploration can be done arbitrarily, however schemes can give a measure of interestingness (the more properties that can be traced back to schemes it fulfils, the more interesting a notion is).

The purpose of our case study is to evaluate the scheme-based exploration model: is it suitable for doing what it claims, can the explorer "invent" significant results? In this paper we showed that indeed this is possible, the idea of decomposition leads to the "invention" of the quotient-remainder theorem, and of the prime decomposition theorem. In fact we compare the theory generated to a well-known textbook [9], and the results are encouraging.

[3]We want to distinguish from *formalization* as taking a fixed theory development path (e.g. a math book) and proving/verifying it in some system.

Status of the Implementation. We used THEOREMA to carry out the case study: all the schemes presented are formulated in the system. We have a prototype mechanism to handle scheme instantiations. However, it is the user who has to provide the appropriate substitutions. This can be further improved by automating the choice of possible instantiations from the language (signature) of the theory.

Most of the proofs involved in our case study are done automatically using the available provers of THEOREMA. However, in particular the proof of correctness of the irreducible decomposition is still work in progress (the proof in this paper is close to what the system will produce). This is due to the fact that this needs underlying multiset theory (including inference support), which is lacking at the moment.

The management of the exploration process is done (manually) by the user. Tools to assist this process (query, knowledge retrieval, transport, etc.) would greatly help the exploration process.

While we are not at a stage where all the aspects of theory exploration are supported by the system, we believe that case studies such as the one presented in this paper, which are in part "pen-and-paper" can help achieve the goal of full computer support: they help the design of provers, theory exploration tools, specifications for theory exploration frameworks.

Scheme Based Insight. Arguably, somebody who studies decomposition domains such as described in this papers, already knows about natural numbers. Note, however, that Buchberger's exploration model provides a methodology for exploring/inventing mathematics. Once the user is familiar with the exploration mechanism, and with a few mathematical ideas (schemes), (s)he can explore freely many theories. Remember that many mathematics courses take students on fixed paths, with often little room for experimentation. We believe that the scheme based model has a great didactic value, and it should be pursued both as a research and teaching tool.

References

[1] Asperti, A., Buchberger, B., Davenport, J., editors: *Mathematical Knowledge Management: Sec-*

ond International Conference, MKM 2003, Bertinoro, Italy, February 16-18, 2003, volume **2594** of *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, 2003.

[2] Benzmüller, C.: The CALCULEMUS Research Training Network: A Short Overview. In *Proceedings of the 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (CALCULEMUS 2003)*, pages 1-16, Rome, Italy, 2003. MMIII ARACNE EDITRICE S.R.L.

[3] Buchberger, B.: Algorithm–Supported Mathematical Theory Exploration: A Personal View and Strategy. *Lecture Notes in Artificial Intelligence, Springer*, 7th Conference on Artificial Intelligence and Symbolic Computation(Research Institute for Symbolic Computation, Hagenberg, Austria) (Proceedings of AISC 2004):16, September.

[4] Buchberger, B., Crăciun A., Jebelean T., Kovacs L., Kutsia T., Nakagawa K., Piroi F., Popov N., Robu J., Rosenkranz M., Windsteiger W.: Theorema: Towards Computer-Aided Mathematical Theory Exploration. *Journal of Applied Logic*, **4**, pages 470-504, 2006.

[5] Colton, S.: *Automated Theory Formation in Pure Mathematics.* Distinguished Dissertations, Springer Verlang, 2002.

[6] Crăciun, A., Hodorog M.: The Quotient–Remainder Theorem for Naturals: Discovery by Lazy Thinking. Technical Report no.06-05, IeAT, 2006.

[7] Hodorog, M., Crăciun, A.: Scheme-Based Systematic Exploration of Natural Numbers. *Proceedings of SYNASC 2006, 8th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing Timisoara, Romania (2006)*, pages 23-34, Timisoara, Romania, September 26-29 2006. IEEE Computer Society Press.

[8] Lenat, D.: *An Artificial Intelligence Approach to Discovery in Mathematics.* PhD thesis, Stanford University, 1976.

[9] Manna, Z., Waldinger, R.: *The Deductive Foundations of Computer Programming* Addison-Wesley Publishing Co, SUA (1993).

[10] McCasland, R., Bundy, A., Smith, P.: Ascertaining Mathematical Theorems. *Electronic Notes in Theoretical Computer Science*, **151(1)**, 2006.

Appendix: Library of Knowledge Schemes for the Systematic Exploration

Independent Knowledge Schemes

Algebraic Knowledge Schemes

$$\underset{p,op,zero}{\forall}(\textit{is-monoid}[p,op,zero] \Leftrightarrow \bigwedge \left\{ \begin{array}{l} \textit{is-semigroup}[p,op] \\ op[x,zero]=x \end{array} \right.) \ ,$$

$$\underset{p,op,zero,inv}{\forall}(\textit{is-group}[p,op,zero,inv] \Leftrightarrow \underset{p[x]}{\forall} \bigwedge \left\{ \begin{array}{l} \textit{is-monoid}[p,op,zero] \\ op[x,inv[x]]=zero \end{array} \right.) \ .$$

Relational Knowledge Schemes

$$\underset{p,r}{\forall}(\textit{is-preorder}[p,r] \Leftrightarrow \underset{p[x,y,z]}{\forall} \bigwedge \left\{ \begin{array}{l} r[x,x] \\ (r[x,y] \wedge r[y,z]) \Rightarrow r[x,z] \end{array} \right.) \ ,$$

$$\underset{p,r}{\forall}(\textit{is-partial-ordering}[p,r] \Leftrightarrow \underset{p[x,y]}{\forall} \bigwedge \left\{ \begin{array}{l} \textit{is-preorder}[p,r] \\ (r[x,y] \wedge r[y,x]) \Rightarrow x=y \end{array} \right.) \ ,$$

$$\underset{p,r}{\forall}(\textit{is-strict-partial-ordering}[p,r] \Leftrightarrow \underset{p[x,y,z]}{\forall} \bigwedge \left\{ \begin{array}{l} \neg(r[x,x]) \\ (r[x,y] \wedge r[y,z]) \Rightarrow r[x,z] \\ r[x,y] \Rightarrow \neg(r[y,x]) \end{array} \right.) \ ,$$

$$\underset{p,r}{\forall}(\textit{is-strict-total-ordering}[p,r] \Leftrightarrow \underset{p[x,y]}{\forall} \bigwedge \left\{ \begin{array}{l} \textit{is-strict-partial-ordering}[p,r] \\ r[x,y] \vee r[y,x] \vee x=y \end{array} \right.) \ .$$

Knowledge Schemes Dependent on the Theory of Natural Numbers

$$\underset{f,g,h}{\forall}(\textit{is-rec-nat-binary-fct-1l}[f,g,h] \Leftrightarrow \underset{is\text{-}nat[x,y]}{\forall} \bigwedge \left\{ \begin{array}{l} f[0,y]=g[y] \\ f[x^+,y]=h[f[x,y]] \end{array} \right.) \ ,$$

$$\underset{f,g,h}{\forall}(\textit{is-rec-nat-binary-rel-2}[f,g,h] \Leftrightarrow \underset{is\text{-}nat[x,y]}{\forall} \bigwedge \left\{ \begin{array}{l} f[x,0] \Leftrightarrow g[x] \\ f[x,y^+] \Leftrightarrow (h[x,y] \vee f[x,y]) \end{array} \right.) \ ,$$

$$\underset{r,q,s,const}{\forall}(\textit{is–nat–step–recr–rel–0–1–1}[r,q,s,const] \Leftrightarrow \underset{is\text{–}nat[x,y]}{\forall}(r[x,y] \Leftrightarrow \left\{ \begin{array}{ll} const & \Leftarrow y=0 \\ q[x,y] & \Leftarrow x>y \\ s[r[x,y-x]] & \Leftarrow \text{otherwise} \end{array} \right.)).$$

$$\underset{f,g,h}{\forall}(\textit{is–nat–step–recl–fct–1–1}[f,g,h] \Leftrightarrow \underset{\substack{is\text{–}nat[x,y] \\ y>0}}{\forall}(f[x,y] = \left\{ \begin{array}{ll} g[x] & \Leftarrow x<y \\ h[f[x-y,y]] & \Leftarrow \text{otherwise} \end{array} \right.)),$$